Paper No. AIAA-2001-4717

ABSTRACT

AN INTRODUCTION TO THE GILGAMESH PIM ARCHITECTURE

Thomas Sterling and Larry Bergman
Jet Propulsion Laboratory
Pasadena, California

Objective

Throughout the history of computer implementation, the technologies employed for logic to build ALUs and the technologies employed to realize high speed and high-density storage for main memory have been disparate, requiring different fabrication techniques. This was certainly true at the beginning of the era of electronic digital computers where logic was constructed from vacuum tubes and main memory was produced by wired arrays of magnetic cores. But it is also true with today's conventional computing systems. Yes, both logic and memory are now fabricated with semiconductors. But the fabrication processes are quite different as they are optimized for very different functionality. CMOS logic pushes speed of active components while DRAM storage maximizes density of passive capacitive bit cells. As a consequence of this technology disparity between the means of achieving distinct capabilities of memory and logic, computer architecture has been constrained by the separation of logical units and main memory units. The *von Neumann bottleneck* is the communication's channel choke point between CPUs and main memory resulting from the separation imposed by these distinct device types. Much of modern microprocessor architecture is driven by the resulting data transfer throughput and latency of access due to this separation as well as the very different clock speeds involved. More subtle but of equal importance is the limitations imposed on the diversity of possible structures that might be explored and achieved were it feasible to bridge this gap and intertwine memory and logic. An entire class of computer structure design space has been precluded because of this technological barrier. Content addressable memories, cellular automata, systolic arrays, neural networks, and adaptive computing structures have all been limited in their practicality and application because of the isolation of DRAM cells from CMOS logic. And without the means of embedding logic in memory, many other structures not yet conceived will never be investigated let alone practically applied to real-world problems.

Several years ago, semiconductor device manufacturers developed a new generation of process and established fabrication lines that allowed for the first time the design and implementation of single chip components integrating CMOS logic with DRAM cells. The technology barrier between logic and memory was eliminated. For the initial processes, some compromises were necessary: device density and logic speeds were not as high as the best segregated technology wafers of the time. Gate switching rates were approximately a factor of 2 or more slower. But many other advantages were incurred by this breakthrough in manufacturing processes. Since then, second generation lines have been installed with the speed-density disparity shrinking significantly. An entirely new digital component design space has been enabled. Two classes of structures made possible by the merger of logic and memory are System On a Chip (SOC) and Processing In Memory (PIM). SOC is a direct porting of classical system configurations including processor core, L1 and L2 caches, memory and I/O busses, and DRAM main memory on to a single chip, thus exploiting a number of advantages incurred through these new fabrication processes. PIM extends the design space much farther by closely associating the logic with the memory interface to realize innovative structures never previously possible and thus exposing entirely new opportunities for computer architecture. It is concepts of this second class of computing organization that is the focus of the work conducted under the Gilgamesh project and described in this paper.

The ability to co-locate and integrate CMOS logic and DRAM cell arrays on the same die provides the

potential for an unprecedented degree of coupling between these two historically segregated digital devices. A number of advantages compared to conventional practices are implied by this new strategy to devising digital structures. To what degree they are exploited depends on the specific architecture devised and the operational execution model employed.

Memory Bandwidth

A memory access operation to a DRAM cellblock acquires an entire row of 1K or more bits in a single cycle. Ordinarily, only a small number of the bits (in the worst case, only one) are used per access in conventional systems as one or a few are selected from the contents of the row buffer to be deposited at the memory chip's output pins. PIM positions logic directly adjacent to the row buffer exposing the entire contents of an acquired memory row. PIM also permits the partitioning of the chip's memory contents in to multiple separate memory blocks, each operating independently. Although smaller in capacity than the entire chip's storage contents, each of these blocks has the same row length, thus increasing the internal peak memory bandwidth proportional to the number of memory blocks. Employing today's fabrication processes, a typical PIM chip could deliver a peak on-chip memory bandwidth on the order of 100 Gbps or more. On moderate scale array of MIND chips can exceed one Tera bytes per second bandwidth.

Access Latency

The close proximity of the PIM logic to the cell block memory row buffers permits short transit times of the acquired data to the processing ALU. Unlike conventional system architectures, there need not be multiple levels of memory hierarchy comprising one or more caches between the processor and memory in combination with the cache control logic delays. Nor is there the multiple stages of communication required between each level and the resulting propagation delays. While delays may vary widely, the degree of improvement can be a factor of two or more. Additional delays incurred due to contention for shared communication channels are also avoided since such accesses are local to a specific chip and do not require system level resources.

Efficiency in the Presence of Low Locality

Most modern memory systems supporting high-speed microprocessors employ a hierarchy of SRAM caches that rely on temporal and spatial locality of data access to operate efficiently. Temporal locality is the property of data usage that reflects the tendency for multiple references to the same word within a narrow time frame. While many classes of applications work well within this framework, a number of important types of algorithms exhibit low or no temporal locality in the short term making poor use of cache resources and resulting in low processor efficiency. Among such cache unfriendly operations include the manipulation of irregular data structures, pointer chasing, parallel prefix, and gather scatter. PIM makes it possible to perform low temporal locality operations directly in the memory without experiencing the long transit times or cache disruption. Such data intensive functions can be performed in place without any data movement off chip and performed simultaneously across an array of PIM chips, yielding very high efficiency with respect to conventional systems undertaking equivalent tasks.

Low Gate Count Processors

Within a multi-chip PIM structure, performance is dominated by effective memory bandwidth and not ALU utilization, as is the case with conventional microprocessors. The design considerations for PIM processors can differ greatly from those of typical processors emphasizing availability to servicing data from memory rather than high floating-point throughput. In contrast to current trends in processor design, PIM processors can be implemented with a relatively small gate count. Since they operate directly on data from their immediate memory, data caches are of less importance and may be eliminated entirely in some cases. Under certain conditions, execution control can be simplified as well. As a result, PIM processors may be implemented in a few hundred thousand devices rather than many millions. This provides simplicity of design for rapid development and high confidence as well as contributing to other features discussed below.

Low Power Consumption

A major advantage of the PIM concept is its excellent power consumption efficiency. This is a consequence of several aspects of PIM structure and operation that distinguishes it from conventional processor design practices. One important factor is reduced use of external chip input and output drivers. Data transfers through IC pins is a major source of power consumption for ordinary systems, especially for high speed I/O. Driving transmission lines and buses can consume a significant portion of a systems total power budget. But PIM performs many of a system's

operations on chip, avoiding the necessity of moving the data to the caches and registers of some remote processor and therefore eliminating the pin driver usage for those operations. Another improvement in power consumption is derived from the reduction in gate count for the processors. Historically, the trend has been reduced efficiency per transistor with respect to performance. As transistor count has increased, the concomitant performance gain has not been proportional and power consumption has exceeded a hundred watts for many processors. By reducing the number of devices per processor by an order of magnitude, the amount of power consumption per operation performed is greatly reduced as well. The reduction or elimination of caches is one source of such improvement. Today, caches can cover as much as three quarters of a microprocessor chip's area and consume considerable power. PIM low dependence on cache structures diminishes its power budget substantially. Redundant memory accesses are also a contributor to power consumption. Because conventional remote processors rarely acquire the data contents of an entire memory block row (some new memory chips are improving this aspect of system operation), multiple read access cycles to the same row are often required. PIM memory exposes all row data to their local processors, permitting a single access cycle to suffice and reducing the total power consumed to affect the equivalent data availability to processing logic.

High Availability

PIM itself is not intrinsically fault tolerant. But PIM exhibits a number of properties that are conducive to realizing high availability architectures. The multi-nodal organization of a PIM chip provides a natural boundary of repetitive structure that can be exploited for reliability. Since each node is capable of carry out a computation independently, it is able to take on work that was to be performed by a failed node on the same chip. The overall performance of the chip is reduced but its basic functionality is retained in the presence of faults. In this mode of operation, high availability is achieved through graceful degradation. Faults may be transient such as single even upsets or permanent such as hard faults. PIM allows one node to diagnose another, and if the failure mode is ephemeral to correct the error and bring the faulty node back online. If the failure is a hard fault, then the node in question can be isolated from the remaining chip units by the other nodes, permitting continued operation of the chip. Many other issues remain before true nonstop computation can be realized. But PIM clearly is beneficial to accomplishing this goal.

In spite of these attributes, Processor-in-Memory technology has been slow to migrate in to commercial computing products. With the exception of limited penetration in the stand-alone embedded computing market (e.g. Mitsubishi M32R/S) and research into data streaming accelerators (e.g. UC Berkeley IRAM), PIMs potential has gone largely untapped. There are multiple factors contributing to this lethargy in exploiting the potential opportunities. One of course, is that such usage is outside the scope of common system practices and therefore must compete with the inertia of an installed base of hardware and software products addressing similar user application base. But PIM architecture, while enticing, has proven inadequate to the promise and challenges of realizing effective general PIM-based computing. In spite of ten years or more of development, PIM has not significantly penetrated the high-end market. The reasons relate to the limited generality of extant chips, the challenge of integrating them within otherwise conventional systems, and the inadequacy of current programming methodologies as well as resource management techniques. The Gilgamesh project is developing the MIND architecture, an advanced PIM-based scalable building block that addresses many of these challenges to a significant degree.

Gilgamesh System Architecture Overview

The Gilgamesh architecture is developed in the context of the new structural and operational opportunities made possible by PIM technology. It is designed to support high performance computing both for spaceborne platforms and ground based supercomputers. The unique properties of PIM allows new structures and capabilities within memory devices previously impossible. Gilgamesh extends PIM computing from basic physical level to virtual level naming and addresses of both data and tasks. It provides hardware support for message driven (parcel) computation and multithreaded control of local execution. The architecture is developed to provide a basis for fault tolerance support and real time processing. It is intended to operate at low power compared to conventional systems while provide great scalability to meet many different system requirements.

Three Primary Levels

The Gilgamesh system architecture exhibits a hierarchical structure of functional elements and their

interconnection. Different system implementations may vary dramatically in their actual structure depending on scale, functionality, and relationship to other elements of the global system in which they are embedded. Nonetheless, all Gilgamesh systems may be devised within a three-level framework as described below.

System Level Organization

The top level of the Gilgamesh system architecture is defined in terms of the number of MIND modules employed, their interconnect topology and network components, and the external devices attached to it. In principle, a Gilgamesh may be as small as a single MIND chip or as large as a three-dimensional mesh incorporating thousands of such chips. At this top level, the integrated MIND modules may comprise a standalone system or they may be part of a larger system with external microprocessors, one or more levels of cache supporting these processors, and mass storage such as disks for persistent backing store. At this level, a Gilgamesh system may be a parallel embedded processor connected to sensors and controlling computers to further process their result data or it may be a general purpose computer connected to user I/O devices and external internet connect.

MIND Chip Level Subsystems

The MIND module or chip has an internal structure that includes memory, processing, and communication functionality. It is capable of fully independent operation or as a cooperating element in a highly parallel structure. The MIND module incorporates a number of processor/memory nodes that store the internal data and control the system operation. It also includes shared functional units such as floating point processing used by all of the nodes. The MIND chip has several external interfaces to support its integration as part of larger systems. The parcel interface supports interaction among the MIND modules making up a Gilgamesh architecture. An external master-slave interface allows the MIND module to be used under the control of external microprocessors or so that the MIND module can control external I/O or mass storage devices. A streaming interface permits direct memory access of large blocks of data at high speed such as data from mass storage or image sensors. Separate signal lines permit rapid response to external conditions and the control of external actuators.

MIND Node Architecture

The processor/memory node architecture all functionality required to perform core calculations and manage physical and logical resources. Each node comprises a memory block of a few Mbits of data organized in rows of 1 Kbits or more and accessed in a single memory cycle. The node processor architecture differs substantially from conventional microprocessors, emphasizing effective memory bandwidth instead of ALU throughput. The node ALU is as wide as the row buffer and can perform basic operations on all row buffer data simultaneously. A wide-register bank permits temporary storage of the contents of row buffer. A multithreaded scheduling controller supports the concurrent execution of multiple separate threads, simplifying management of resources and handling hazards. Each node interfaces with others on the chip as well as with external interfaces by means of a common shared internal communications channel.

Modes of System Integration

The MIND architecture and chip design are devised to address the requirements of a diversity of advanced system requirements. It may contribute to a wide range of operational contexts from simple single-chip embedded computing tasks to large hierarchical Petaflops-scale supercomputers and many configurations in between. MIND chips may perform as masters, slaves, or in peer-to-peer relationships. They may operate alone, within a homogenous structure comprised uniquely of themselves, or in conjunction with a plethora of other component types including other technologies. They may be completely responsible for all aspects of a computation or provide specific optimal but narrow mechanisms contributing to the broader computing model supported and even guided by other elements as well. Depending on their role, the organization of the systems that they in part comprise may vary. MIND is of a sufficiently general nature that the ways in which it may be employed is larger than it is reasonable to describe exhaustively in this note. Some key classes of structure are briefly discussed to suggest the manner and means of their utility.

Single-chip embedded

The simplest system employing the MIND component is a single chip structure in which all responsibilities of computation and external interface are supported by the one device. The chip

interfaces permit independent input and output signals for sensors and actuators, a control bus for managing external slaved devices such as secondary storage and user interfaces, and a data streaming port for rapid transfer of bulk data such as that from real time image (CCD) sensors. Although a single chip, a MIND device still incorporates multiple processor-memory nodes to provide mutual fault diagnosis, high performance through parallel computing, bounded real-time response, and graceful degradation in the presence of faults.

Gilgamesh scalable

Gilgamesh is a scalable system comprising multiple MIND chips interconnected to operate as a single tightly coupled parallel computer without additional processing support. The number of MIND chips within a Gilgamesh systems can range from a few (typically four to sixteen) that easily fit on a small to moderate board to extremely large systems of many thousands or even a million chips packaged possibly in a 3-D matrix. A cubic structure of MIND chips a meter on a side, including cooling could sustain Petaflops scale computation. The actual interconnect network and topology will differ depending on Gilgamesh system scale and time critical factors as well as power and reliability considerations. The array of MIND components shares a global virtual name space for program variables and tasks that are allocated at run time to the distributed physical memory and processing resources. The MIND chips interoperate through an active message protocol called *parcels* that supports everything from simple memory access requests to the remote invocation of entire programs with efficient light-weight transport, interpretation, and context switching mechanisms for effective handling of a range of parcel packet sizes. Individual processor-memory nodes can be activated or powered-down at run time to provide active power management and to configure around faults.

Smart memory – slaved

MIND chips can be a critical component of larger systems, replacing the "dumb" memory of a conventional system with smart memories capable of performing operations within the memory chips themselves. Such systems still maintain the conventional structure involving one or more microprocessors responsible for conducting, coordinating, and managing the computation and overall system resources performing it. Such a structure may even employ a typical cache hierarchy for SMP or DSM operation, or support shared memory without cache coherence. The MIND chips replace some or all of the main memory in such structures, providing a memory system with logic for local operations.

In its most simple form, MIND employed as smart memory can be used directly in place of conventional DRAM (or advanced versions) parts. This does not mean they would plug in to the same slot; pin compatibility is unlikely. But the general structure remains identical, even if the pin-outs are modified. All MIND chips are operated directly under the control of the host or main microprocessors of the system. In this slaved mode, the MIND components receive direct commands from their host microprocessors. These may be as simple as basic read-write requests to the memory itself or compound atomic operations such as *test-and-set*. But the set of operations that can be performed is much larger and in slaved mode MIND chip array can perform a wide array of such instructions on streams of physical memory blocks. Performing scaling functions for numeric applications or associative operations for relational transaction processing are two such examples, each operation triggered by a command from the host microprocessor but performed in parallel by the many processor-memory nodes on the array of MIND chips. This data parallel operational mode can be extended to the execution of simple multi-instruction functions, invoked by the host processor(s) and performed on local data by each of the MIND nodes. In slaved systems, all MIND chips, like their dumb DRAM counterparts, are interconnected through the system memory channel.

Smart memory – peer to peer

There are many opportunities to derive performance benefit through the execution of data parallel instructions or functions on single data elements or contiguous blocks of physical memory by means of a master-slave relationship described above. This is the primary way by which the majority of PIM architectures are structured and their computations managed. However, more sophisticated functions require access to data that may be distributed across multiple nodes, not just local to a particular memory block. One important class of operations involves irregular data structures that incorporate virtual pointers that must be de-referenced by the MIND nodes

themselves identifying data values stored on other nodes. In other cases, the result of a function performed on one node may require an intermediate solution to a computation performed on another node. Under such circumstances, more general computations than those accomplished in a slaved mode require a peer-to-peer relationship among all of the MIND nodes. This could be achieved through the common memory channel with provision made either for the host microprocessors to coordinate and conduct transactions between chips or for the MIND chips themselves to become master of the memory channel. However, traffic congestion due to contention for this shared communication resource would impose a bottleneck that could severely constrain throughput and scalability. Rather, where peer-to-peer inter-MIND chip operation and virtual memory references are to constitute a significant proportion of the system behavior, a richer network infrastructure dedicated to communications between MIND chips is to be preferred, altering and extending the structure beyond that of conventional systems. The MIND architecture supports the use of an independent memory network and node-to-node direct cooperation for peer-to-peer functionality. The host microprocessors are still responsible for the overall control of the computation and broad coordination of the system resources. But at the finer grained details of the system operation, in a peer-to-peer functional relationship, the MIND nodes themselves interoperate on behalf of but not in direct control by their hosting microprocessors.

Smart memory – master through percolation

An innovative concept has emerged from the HTMT project that may revolutionize the relationship between processors and their main memory, enabled by the potential of advanced PIM architecture such as MIND. Traditionally, in addition to performing the actual computations related to an application for which the conventional register-register microprocessor architectures are optimized, they also are required to synchronize the tasks making up the computation and manage the movement of instruction, variable, and context data to high speed local memory, usually L1 and L2 caches. Because of latency and the classical memory bottleneck, load store operations are not very effective and can cause significant reduction of processor efficiency. PIM provides a new opportunity to significantly improve the efficiency and scalability of shared memory MPP systems and the MIND architecture is devised in part to implement the new paradigm. *Percolation* is a proposed methodology by which the control of physical and logical components of a computation are managed not by the main microprocessors that are ill suited to these responsibilities but to the main memory incorporating PIM MIND chips. Under the percolation model the small, inexpensive, and highly replicated MIND processors assume the task of managing all memory resources, accomplishing all memory intensive functions (such as parallel prefix, or associative update), and coordinating the execution distributed parallel tasks. Most importantly, percolation provides the means of migrating all necessary data to the local high speed memory or cache of the main microprocessors and scheduling their use; thus relieving the hosts of all overhead and latency intensive activities. Through percolation, the smart memory can become the master and the main microprocessors become the slaves revolutionizing the architecture and operation of future generation high performance computers.

Interconnect

MIND modules are connected by a fabric of channels that permit parcel message packets to be passed between any pair of nodes within a Gilgamesh system. The topology of the Gilgamesh interconnect may differ depending on the system scale, usage, and requirements. The parcel transport layer hardware interface supports various network structure through a programmable routing table. Each MIND module contains multiple parcel interfaces. For example, there may be on parcel interface for each MIND node on the chip. Through the internal MIND module shared communication channel, incoming parcel messages can be rerouted to other parcel interfaces. The MIND chip can act as a routing switch of degree equal to the number of parcel interfaces on the chip. With four parcel interfaces on each chip, a small four chip system has complete interconnect. A sixteen chip hyper-cube interconnect can be implemented with the same chip type. Mesh and toroidal structures can be implement for larger organizations as well. For higher bi-section bandwidth and shorter latencies for larger systems, external networks comprising independent switches may be used to build more powerful networks. These can be particularly useful when MIND chips are used in conjunction with larger systems.

MIND Module Subsystems

The principle building block of the Gilgamesh systems including the anticipated PIM enhanced main memory subsystems of future high performance computers is the MIND chip or module. The MIND module is designed to serve both as a complete standalone computational element and as a component in a synergistic cooperation with other like modules. The subsystems comprising the MIND module are devised to support both its own internal functionality and its cooperative interrelationship with other such devices. This section provides a brief description and discussion about the chief critical subsystems making up a MIND module.

MIND Nodes

The MIND node is the principal execution unit of the MIND architecture. Multiple nodes are incorporated on a single MIND chip, the exact number dictated by fabrication technology, chip real estate, and design considerations including the number of gates per node processor. The node consists of the node memory block, the node wide-word multithreaded processor, and the connections to parcel message interface and the MIND chip internal bus. The MIND node memory is DRAM with the entire row buffer exposed to the node processor. The node processor ALU and data paths are structured to make the best usage of the high bandwidth direct access to the memory block. A wide register bank is integrated in to the data path. Each wide-register is the width of the row buffer and allows temporary buffering of the contents of the row buffer. The ALU is also capable of working on all bits of the row buffer or wide register within one memory access cycle time. The node executes instruction streams called threads. A thread employs a wide-register as its primary state definition. A multithreaded sequencer manages multiple threads concurrently, allowing interleaved instructions from among the many active threads to share the same physical execution resources, actually simplifying handling of data and control hazards as well as providing rapid response to real time signals. Node memory is used to contain pages with virtual addresses and address translation is performed among the nodes through a distributed address mapping directory table.

Shared Function Units

While each node incorporates essentially all logic needed to undertake any general computation, some additional units can extend the operational capability of the nodes while not necessarily justified for inclusion within each and every node. Not every possibly functional unit may have sufficient usage to warrant replication on a single chip or may require too much die area for more than one such unit to be practical on a given chip. MIND provides the necessary logical and physical infrastructure to permit the addition of separate functional units that can be accessed by all of the MIND nodes as well as through the master-slave external interface. For the first design, three possible such units are under consideration for incorporation: floating point multiply, floating point addition, and permutation network. These can be pipelined, supporting multiple requests concurrently and have their own dedicated access arbitration controllers. Future designs may include additional shared functional units.

Internal Shared Communications

The majority of node operations employ local resources within the node, but some operational functionality is provided through subsystems on the MIND module but external to the specific node. The shared function units described above are examples as are the external interfaces to be described below. Another important class of resource to which every node must have access is the combined memory blocks of the other nodes on the same MIND module. To support the sharing of function units, control of external interfaces, and access to chip-wide memory blocks, an internal shared communication mechanism is incorporated as part of every MIND module. This channel may take any one of several possible forms but provide fast reliable access to all needed chip resources. It is anticipated that such shared communications within the module will employ a split transaction protocol to decouple the communication throughput from the natural operating speeds of the shared elements. Depending on the number of nodes within a module and their speeds, either multi-drop buses or point-to-point network topologies may be employed. But in either case, redundancy or graceful degradation of path between any two subunits within the module is required for fault tolerance. The internal shared communications medium will support its own access arbitration and error detection mechanisms.

Master-Slave External Interface

A Gilgamesh ensemble of MIND units may operate as an independent system or in cooperation, support, or control of external computing elements. The MIND chip architecture incorporates an external interface that

services the necessary communications, command, and control functions for interoperability with external computing components (not including other MIND chips). One or a collection of MIND modules may be slaved and responsive to the commands of one or more external master microprocessors. This would be a comparable relationship to that of a primary microprocessor to its main memory chips except that the MIND chip surrogates can also perform in situ operations. The MIND modules may also perform as master by means of this external interface controlling external devices such as network ports and mass storage devices or real time sensors. In this mode, it may be used in conjunction with the streaming I/O interface described below.

Streaming I/O External Interface

The external streaming interface provides a direct high bandwidth connection between external remote devices and the MIND memory blocks. The interface will support full direct memory access rate of data transfer in or out of the chip. It can be used for such input devices as real time digital cameras or output stereoscopic projectors at full frame rate. Using this interface, MIND units can be used as post sensor processors for digital signal processing tasks such as passive sonar or radar return data. It can be used for accepting large blocks of data from mass storage devices or can dump data in to such devices as holographic storage.

Parcel Interface

Inter MIND chip communications is supported by the parcel packet transport layer. Each MIND chip includes multiple parcel interfaces to an external network providing access to all MIND chip nodes comprising a Gilgamesh system. Parcels have to be fast enough to perform basic memory operations at the same rate that conventional memory chips support memory accesses. Therefore, the interface has to be wide enough to accept the packets for these operations. However, parcels also have to support variable format packets for a wide array of more sophisticated remote operation invocation. Thus, a combination of parallel and serial acquisition of parcel packets are required of the interface. The parcel interface has to be capable of interpreting basic parcel operation types to perform the most simple instructions without demanding full operation of the thread scheduler. For example, a thread can read any state within the node and generate a new parcel containing that state to be returned to the calling MIND chip without employing any higher functionality of the MIND architecture.

Signals

External events from secondary storage devices, sensors, command interfaces (e.g. mouse, keyboard) and other asynchronous triggers can be communicated directly to the MIND module through a set of signal two-state signal pins. Such signal conditions, when detected, can cause an active thread to be immediately scheduled by the multithread controller or cause a new thread to be instantiated and executed. Signal pins can also be used to provide external voltage or current switching to external logic, power, or actuator devices to control their operation. The input and output signal pins are shared among all nodes of the MIND module and can be directly controlled by other MIND modules through the parcel interface or by a master microprocessor through the master-slave external interface.

MIND Node Architecture Overview

The node of a MIND chip provides the primary storage and operational resources of a Gilgamesh system. It manages the DRAM main memory providing both local and remote access to its stored data. It performs basic operations on multiple fields of data simultaneously. It initiates fine grain tasks, carries them out, and completes them, interleaving operations from separate but concurrent tasks to achieve high efficiency. The node assimilates messages and derives new tasks from them. The architecture of the node includes its principal elements, the data paths interconnecting them, and the control logic determining their behavior. This section describes these in some detail.

Memory Block

The node memory block has at its core (no pun intended) one or more conventional stacks of DRAM cells arranged. The stacks are arranged by rows of individual single bit storage cells. Each row may typically contain on the order of 2048 such cells. The sense amps are connected to columns of corresponding cells across all of the rows. However, due to the extra fine pitch of the cells, it is not possible to lay metal output bus lines such that adjacent cells in a given row can be read out at the same time. As a result, groups of eight adjacent cells in a row share the same output bus line and use it in a time multiplexed manner. There are one eighth as many output bus lines as row cells or for

2048 cells per row, there are 256 output bus lines. Thus a row, once addressed, is read in a succession of eight 256-bit groups. The output of the row sense amps is fed directly to the row buffer, a row wide fast register that holds the contents of the row and is available to feed it back to the DRAM cells which is necessary because such a read is destructive. The contents of the row buffer, which represents all the data of the selected row, is then available for immediate processing.

Access to the memory block is managed by the memory controller. This simple hardware subsystem performs two primary functions related to the allocation and operation of the memory block. The memory controller arbitrates among the potential sources of memory requests. These include the MIND intra-chip communications bus, the parcel handler, and the thread coordinator. Access is prioritized with five levels of priority to support both essential accesses and optimal resource utilization. The priorities include:

1. basic
2. preferred
3. exception
4. real time
5. parcel diagnostics

Most thread requests to memory are issued with the *basic* priority, which accounts for the majority of accesses. General accesses through the MIND intra-chip bus from other nodes are asserted with the *preferred* priority as are general memory access parcel requests. As these are shared and limited resources, responding quickly to these requests will free them earlier than would otherwise occur. Supervisor memory requests and interrupt handlers are supported at the *exception* priority level, which takes precedence over the general execution accesses. To support real time operation with bounded and predictable response time, a high priority is provided. The *real time* priority level is distinguished from those below it in that not only will it take precedence over requests at lower priority but it will preempt any current memory request that is being performed, unless the actual discharge phase is taking place. If so, that subcycle will be completed, its contents temporarily buffered and the real time request immediately initiated. The highest priority is *parcel diagnostics* that is used to force the memory controller to operate, even if there is a fault in part of the hardware. This is used when the node has failed and must be controlled externally. In this case, the memory controller is really disabled and the signals provided by the parcel handler.

The memory block itself performs some simple compound-atomic operations on the data stored. Thus the controller accepts as part of the request certain basic op codes to be employed by the wide ALU during such cycles. This allows parcels or other nodes on the MIND chip to perform atomic synchronization primitives without invoking threads and incurring the overhead in space and time that that implies.

Parcel Handler

The Parcel handler is responsible for the transfer of MIND active messages, or Parcels, between MIND modules. Each MIND node has a local parcel handler, although a parcel arriving at any node on a MIND chip can be redirected to any other node on the same MIND chip through the internal intra-chip communications channels. This permits the on-chip parcel handlers to perform as an intermediate router of parcels in a potentially large network of MIND chips.

The parcel carries multiple fields of information that influences its transport across the Gilgamesh system between source and final destination nodes, determines the actions to be performed at the destination MIND node, and specifies the way in which flow control is to be continued upon completion of the parcel task. The basic fields comprising a standard parcel include:

1. target physical node destination
2. context id
3. destination data or object name (virtual or physical)
4. action specifier
5. operands (values or names)
6. continuation

A separate address translation mechanism predicts the physical node on which a virtually named destination is anticipated to be and the parcel moves through one or more hops through the system network to this physical location. At this destination, the presence of the sought after entity is verified. It is possible that the parcel will be redirected to a new physical destination at this point, creating a new physical address in the parcels first field entry. Low-level parcels may deal directly with the system's physical elements for low level diagnostics, initialization, and maintenance including reconfiguration. The context id field indicates domain of execution including physical, supervisor, or any one of a number application name spaces, for hardware supported security. This, in conjunction with the destination name field, fixes the logical (and possibly physical) data or object that the parcel is to effect. The

action specifier dictates that effect. It can be a basic hardware operation performed by the parcel handler on the node. It can be a call to a object method, function, or system service routine that initiates a thread to be executed, or it can contain a sequence of instructions to be directly executed. The operands field is variable length and self formatted possibly containing a mix of different typed values and variable names to be de-referenced during the course of the action to be performed associated with the parcel. When the parcel action terminates, there are two ways to initiate a follow-on action.

Wide Register Bank

The MIND node processor employs a bank of registers, each of which is as wide as the row buffer and sense amps, perhaps a couple of thousand bits, which may be 2048 bits. Being able to repeatedly access an entire row multiple times after the first read cycle is of tremendous value when the cycle time difference exhibited is a ratio of an order of magnitude or more between memory and registers.

Wide ALU

Execution Model and Mechanisms

The Gilgamesh MIND architecture supports a dynamic adaptive resource model of execution at the fine, medium, and coarse levels of granularity. Many of the attributes have been touched upon in earlier sections. The purpose of this brief closing section is to highlight the logical dynamic functionality enabled by the efficient hardware mechanisms of the MIND architecture that contributes to the determination of the allocation of tasks to execution resources both within MIND nodes and shared across MIND nodes and chips. Although there are many ways to manage the node resources, Gilgamesh promotes a dynamic multilevel multithreaded methodology that establishes a new design space for both hardware and software.

The central premise is that unlike conventional multimode statically scheduled systems, the work goes where the data is, rather than always presuming a locus of work sites and designating data partitions local to them. Thus an action or task is to be performed on a dense data construct, possibly multi-field in structure, and a action-specifier is dispatched to the site of the data. This specifier can be tiny compared to the size of the data operated upon. Equally, the virtual data can be distributed by a number of different algorithms and the operation and efficiencies are still retained, whether static or dynamic. This message driven model of computation using parcel type of active messages exploits a simple multithreaded intra-node instruction issue mechanism. For persistant control state, medium grained objects can handle complex timing and control relationships. These points are expanded somewhat below.

Parcel Driven Computing

Parcel dynamic structures were described in some detail in section 4. The important idea is that parcels are an embodiment of a remote task call and a decoupled or split transaction method of creating remote flow control. One consequence of active messages including parcels is their intrinsic property of latency hiding. Once a node has launched a parcel, it can forget about it, using its own resources for other work. Until a remote node assimilates such a parcel, it dedicates no resources to it, doing so only when it has been completely received and interpreted by the node's parcel handler and then only when other tasks are not consuming these resources already. Thus a node can be imagined as a small computing engine that holds data and processes parcels directed to that data while directing result parcels to other data.

Virtual Memory

Unlike almost all other examples of experimental PIM architectures, Gilgamesh manages a distributed but shared virtual memory space such that all data is viewed by all nodes within the system and all address pointers can be translated, perhaps through a mulitstaged procedure so that a parcel will reach the physical location that holds the logical page of data sought. Virtual paged data is used in this structure so that everything within the page is accessed via physical offset to the page header. The distributed address translation scheme creates a distributed page table, the entries of which are allocated to specific physical MIND chips. Actual virtual pages may be placed anywhere but are preferentially placed in the same chip or collection of chips in which the respective directory table entry resides. A new affinity model is offered that allows virtual pages to be realigned to manage locality with out artificially changing the logical data structures themselves. A preparation stage of data realignment, like a kind of gather, will be possible and overlap the

other computation and synchronization phases of execution management for latency hiding.

Multithreaded Control

The basic management of the node memory block was described with several requests for the memory block being asserted at the same time. The multithreaded control unit extends this class of resource management scheme to support other resources such as basis integer and floating point operation units, register to register transfers and perturbations, and others. Such asynchronous instruction issue controllers permit a number of active threads to exists simultaneously, each with a pending action type ready to be performed or waiting for its previously issued instruction to be completed. Although simple in concept, the tradeoffs for optimality are subtle. It turns out that overall performance increases with multithresed systems if the processors are made small and many more nodes are used in order to maximize sustained memory bandwidth which is the critical resource.

Object based computation

Decoupled control.